Data Mining TP3

Deniz Sungurtekin

April 2020

1 Introduction

L'objectif de ce travail pratique est d'appliquer nos connaissances théoriques acquises en cours sur l'estimation de probabilité de densité en utilisant KDE (Kernel Density Estimation). Cette méthode consiste à estimer la probabilité d'une région R donnée depuis n points d'entrainements, dans ce travail j'ai donc utilisé la méthode "univariate" afin d'estimer la fonction de densité, tel que:

$$K_1(x) = \frac{1}{\sqrt{2\pi}} exp(-\frac{1}{2}x^2)$$

représente la "gaussian kernel" permettant de visualiser la différence des valeurs de manière continue.

J'ai donc utiliser cette fonction afin d'estimer la fonction de densité p(x) tel que:

$$p(x) = \frac{1}{nV_R} \sum_{i}^{n} \prod_{j=1}^{j=d} K(\frac{x_j - x_{ij}}{h})$$

Où d est la dimension de nos données, h un paramètre de lissage (taille de notre hypercube, explication en détail dans la section suivante) et Vr le volume de notre région donnée par h^d.

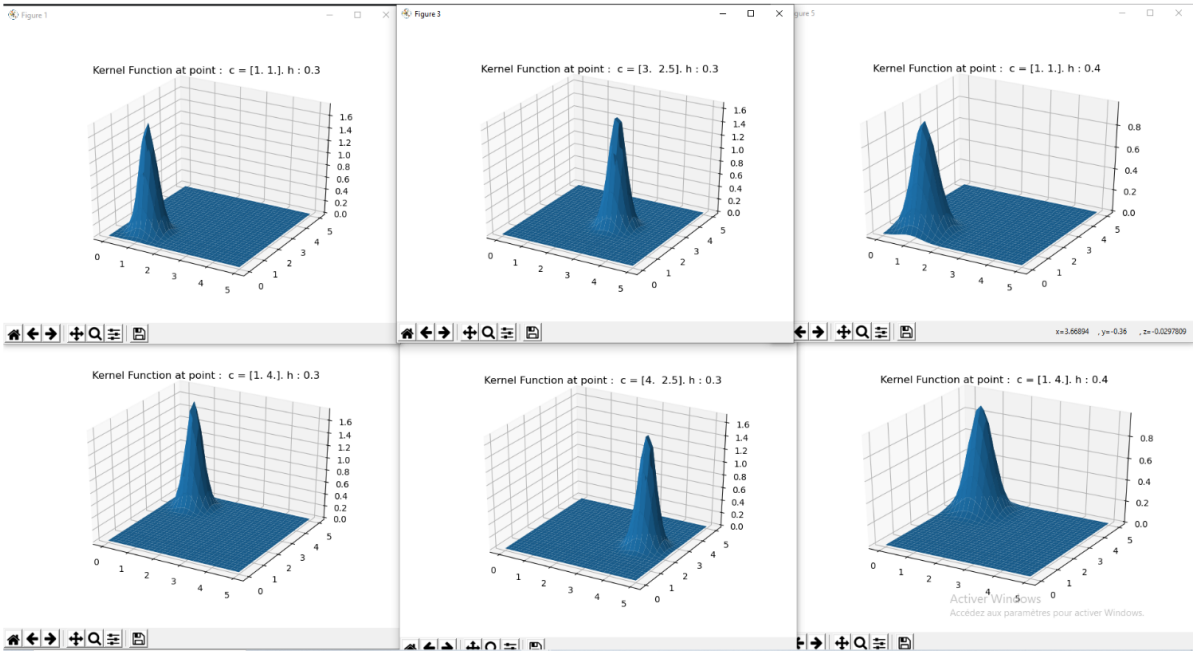
Grâce à cette estimation nous allons donc pouvoir observer l'effet du paramètre h sur un ensemble de donnée trivial et l'effet qu'il engendre sur la méthode de Naive Bayes sur l'ensemble de donné d'Iris.

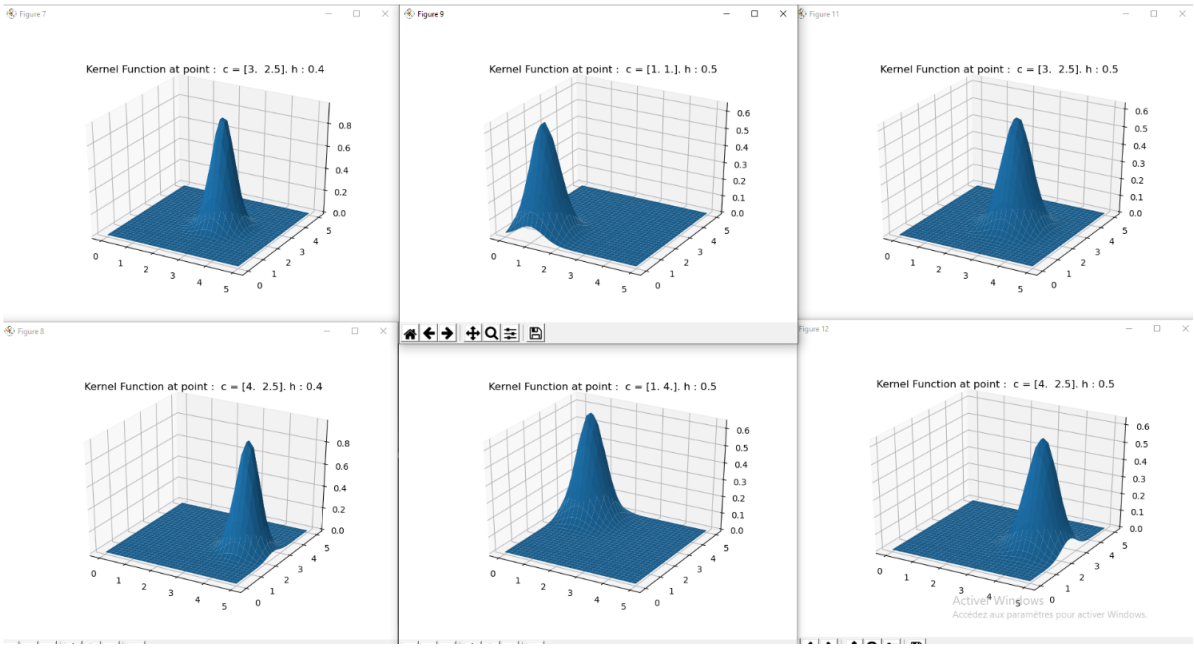
2 Résultat

Exercice 1:

1. Voir la fonction densityEstimation et densityEstimation2 dans le fichier kde.py, la seul différence entre ces deux fonctions est le faite que densityEstimation2 prend en paramètre plusieurs éléments de notre training set c. (La fonction densityEstimation est utilisé uniquement pour calculer p(x|ci)

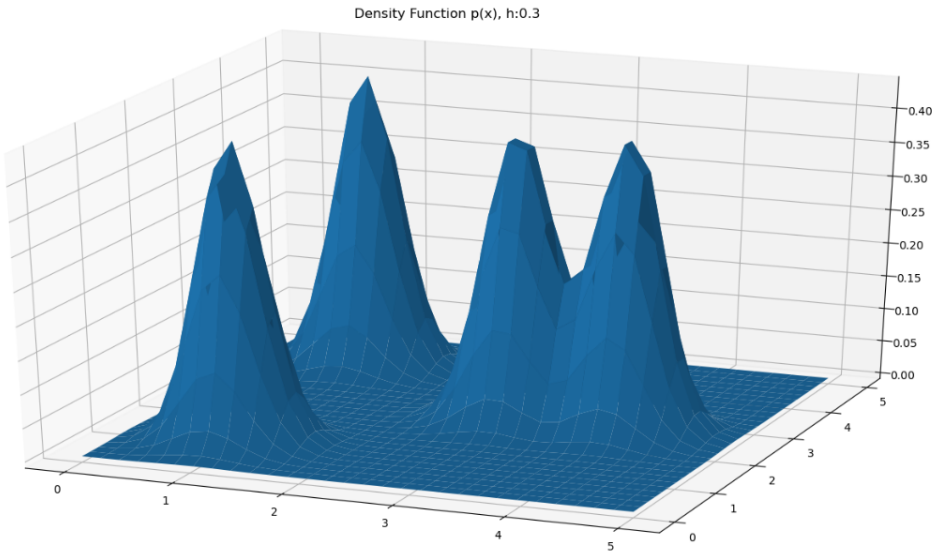
2. Voici mes résultats en images:

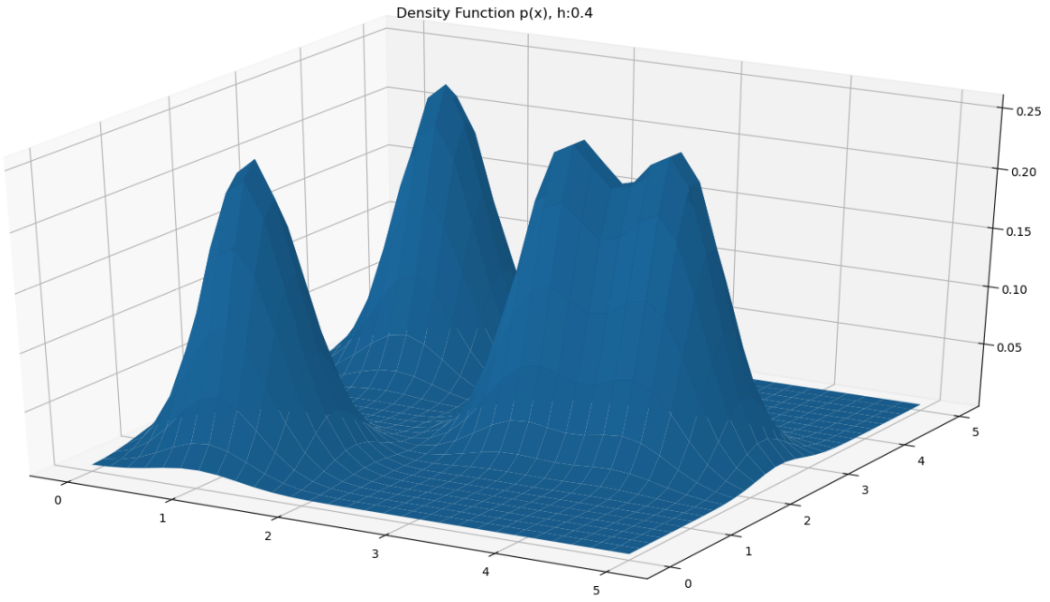


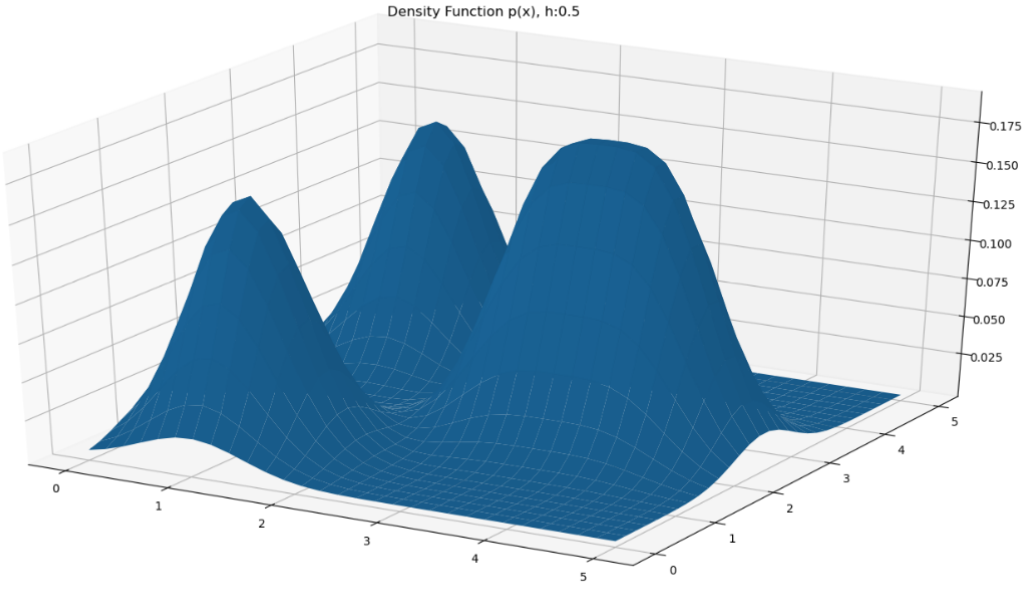


On observe donc que pour chaque figure que le pic est centré au point c puisqu'on soustrait c à x, les seuls endroits possédant une valeur non négligeable sont celles qui sont proche de c. On voit également que la valeur de h influe sur l'épaisseur et la taille de notre cône puisqu'on divise par h notre soustraction x-c et qu'on multiplie notre résultat par un scalaire 1/Vr où Vr dépend de h. Donc plus h est grand plus notre cône est épais et petit.

3. Voici mes résultat pour le calcule de p(x) pour chaque x appartenant à testSet:




Density Function p(x), h:0.4
0.25
0.20
0.15
0.10
0.05
0
1
2
3
4
5

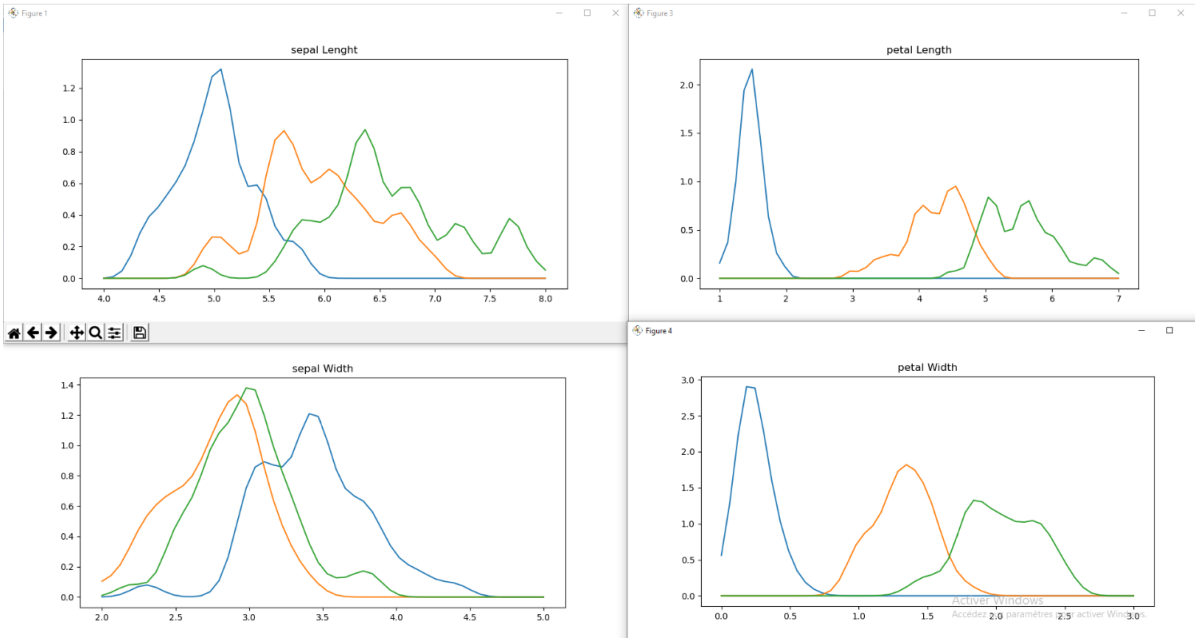

Density Function p(x), h:0.5
0.175
0.150
0.125
0.100
0.075
0.050
0.025
0
1
2
3
4
5

On voit donc ici que lorsqu'on plot l'ensemble de nos cônes que leur épaisseur est importante puisqu'avec h = 0.5 on ne distingue plus les sommets des points [3,2.5] et [4,2.5] étant donné qu'ils ne forment plus qu'un seul cône à eux deux.

Remarque: Mes valeurs sur l'axe Z ne correspondent pas aux vôtres, alors qu'il me semble avoir bien définit ma fonction densityEstimation2.c.f code kde.py

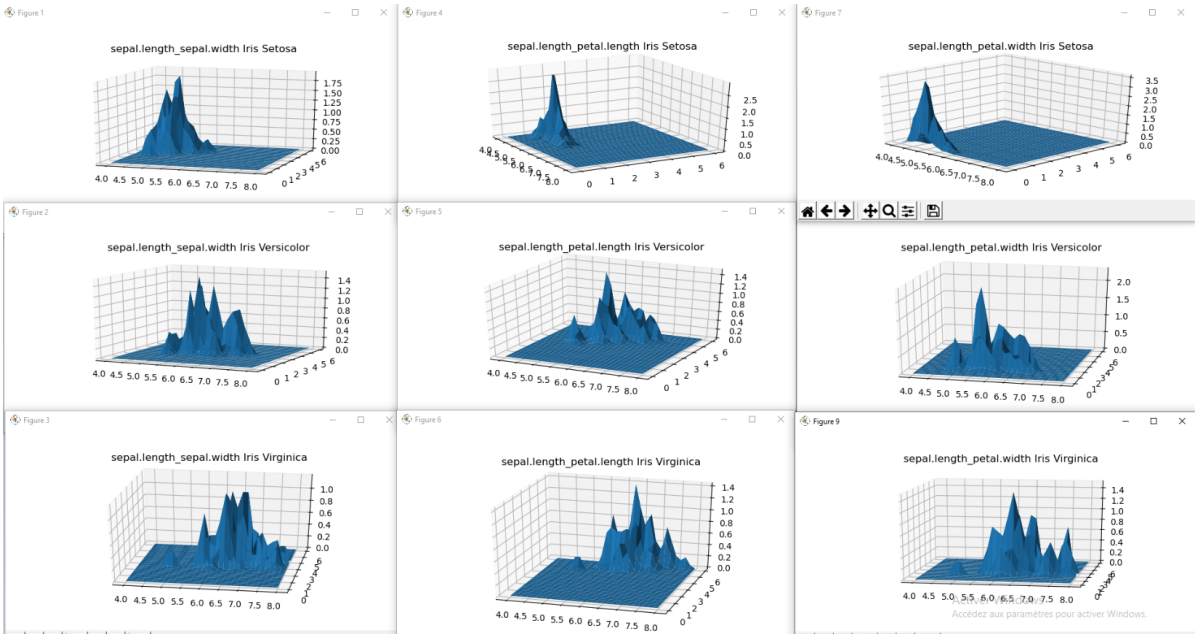
Exercice 2:

1. Voici mes plots pour chaque attributs:



Remarque: La forme de mes fonctions sont les mêmes que les vôtres, cependant leur taille ne correspond pas toujours, je pense que cela est lié au même problème perçu à l'exercice 1 mais sur l'axe Y puisqu'on se trouve en 2D.

2. Voici les plots pour chaque pair d'attributs:



Remarque: Certains plots peuvent sembler être différent, mais cela est uniquement dû à l'intervalle plus grand que j'ai choisit.

3. Lorsqu'on choisit de prendre par exemple h = [0.3,0.4,0.5], on remarque la même chose que pour l'exercice 1, plus h est grand plus la taille est petite (axe Z) et plus les pics s'arrondisent ("smoother") et se confondent.

Exercice 3: Pour l'implementation de Naive Bayes sur Iris j'ai simplement repris la fonction "train" (Sans le meanstd) afin d'obtenir le prior de nos données d'entrainement, ensuite j'ai calculé la fonction de densité pour mon ensemble de donné d'entrainement et de test sur les valeurs des données apparteneant à chacune des classes (pointbyclass[i]) pour i = 0 à 2. Aprés cela, j'obtiens donc mon posterior = likelhood * prior et je prend l'index qui retourne la plus grande valeur.

Avec ceci j'obtiens donc le résultat suivant:

```
TRAIN ACCURACY

Train: with h = 0.3: Got 99 / 100 correct => accuracy: 0.990000
Train: with h = 0.4: Got 99 / 100 correct => accuracy: 0.990000
Train: with h = 0.5: Got 96 / 100 correct => accuracy: 0.960000
TEST ACCURACY

Test: with h = 0.3: Got 48 / 50 correct => accuracy: 0.960000
Test: with h = 0.4: Got 48 / 50 correct => accuracy: 0.960000
Test: with h = 0.5: Got 48 / 50 correct => accuracy: 0.960000

Process finished with exit code 0
```

Dans le TP1, j'avais obtenu une précision de 97/100 pour le trainSet et de 47/50 pour le testSet, on remarque donc une amélioration légère. De plus, on observe que pour le trainSet la précision diminue si h est trop grand.

Remarque: J'obtiens une précision plus grande pour h = 0.1, 100/100 pour le trainSet et 48/50 pour le testSet.

3 Conclusion

Pour finir, on comprend à travers ce travail pratique l'utilisation de l'estimation de fonction de densité (KDE) et la signification de ses paramètres, en particulier le paramètre h qui représente la taille de notre hypercube qui rend notre fonction plus lisse, mais lui fait perdre en précision, si elle devient trop grande comme observé sur Iris et les plots de l'exercice 1 et 2.